SOLID - Liskov Substitution

Programação Orientada a Objeto II

SOLID - Liskov Substitution

Programação Orientada a Objeto II

"Se uma classe A é um subtipo da classe B, então você deve ser capaz de substituir toda ocorrência de B por A mantendo o mesmo comportamento."

Apesar de aparentemente simples, esse pode ser um dos princípios mais complexos dentre os cinco do SOLID.

O ponto de complexidade está no design da aplicação. Se uma classe ou interface foi definida de forma muito acoplada ou restrita, isso pode levar diretamente à violação do princípio da substituição de Liskov.

SOLID - Liskov Substitution

Programação Orientada a Objeto II

Considere uma interface Car com os seguintes métodos definidos:

```
public interface Car {
    void turnOnEngine();
    void accelerate();
}
```

SOLID - Liskov Substitution

Programação Orientada a Objeto II

A interface pode ser facilmente implementada pela classe MotorCar com o seguinte corpo:

```
public class MotorCar implements Car {
    private Engine engine;
    public void turnOnEngine() {
        this.engine.on();
    }
    public void accelerate() {
        this.engine.powerOn(1000);
    }
}
```

SOLID - Liskov Substitution

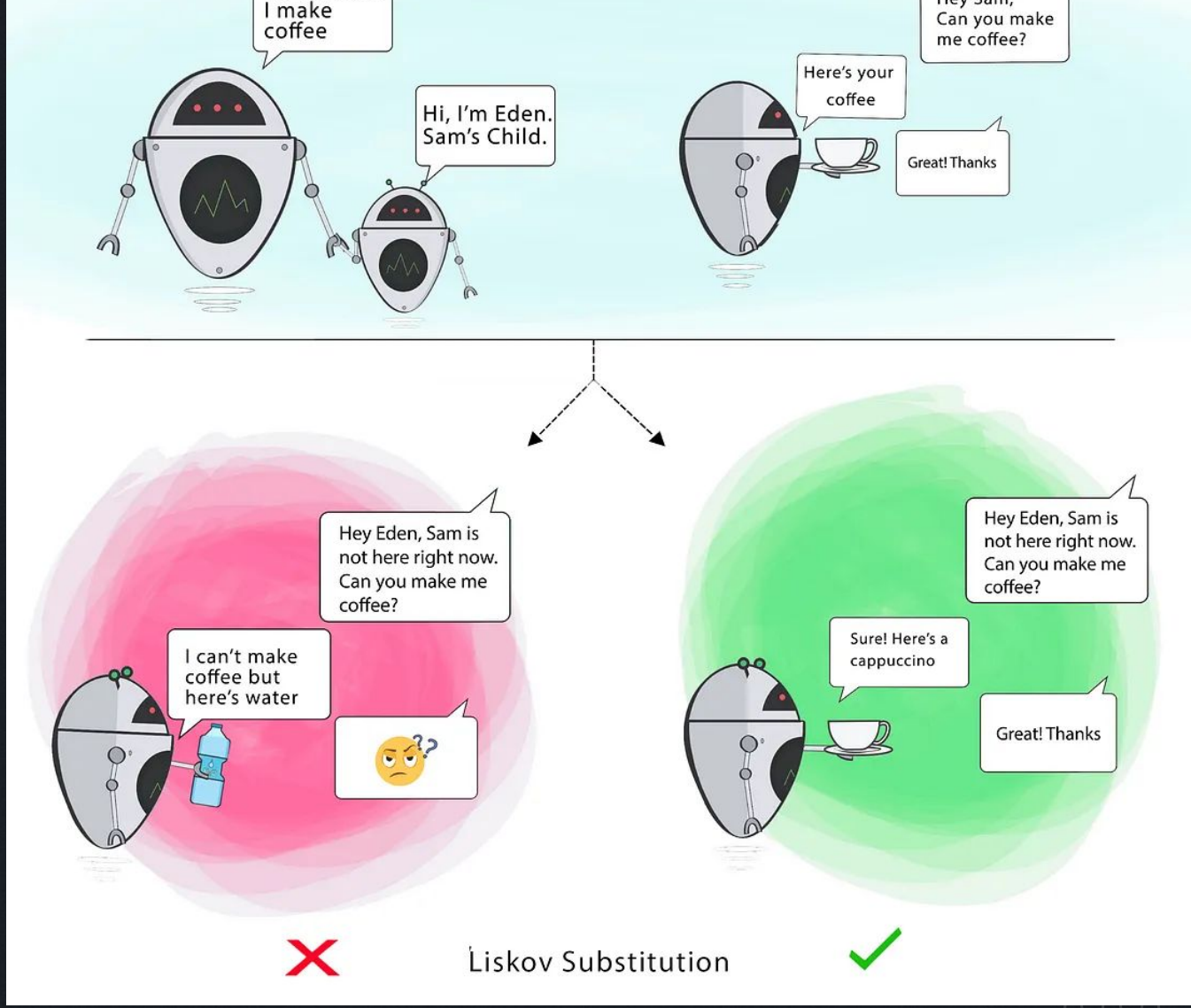
Programação Orientada a Objeto II

No entanto, caso formos representar veículos ecológicos, por exemplo, elétricos, a classe seria um pouco diferente do caso anterior, onde:

```
public class ElectricCar implements Car {
    public void turnOnEngine() {
        throw new AssertionError("I don't have an
engine!");
    }
    public void accelerate() {
        //this acceleration is crazy!
    }
}
```

SOLID - Liskov Substitution

Programação Orientada a Objeto II



Obrigada